Problem 1

This problem is about the Stable Matching problem and the Gale-Shapley algorithm. Suppose we have four men $A, B, C, D$ and four women $1, 2, 3, 4$ with the following preferences (each ordering lists the most preferred choice first):

<table>
<thead>
<tr>
<th>Men: name preference list</th>
<th>Women: name preference list</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1,2,3,4</td>
<td>1: C,B,D,A</td>
</tr>
<tr>
<td>B: 1,4,3,2</td>
<td>2: B,D,A,C</td>
</tr>
<tr>
<td>C: 2,1,3,4</td>
<td>3: C,D,B,A</td>
</tr>
<tr>
<td>D: 2,3,4,1</td>
<td>4: A,B,C,D</td>
</tr>
</tbody>
</table>

a. Consider the following matching: $(A, 4), (B, 1), (C, 3), (D, 2)$. Is this matching stable? Reason your answer. (If the matching is stable, prove it. If it is not, specify a pair that prefers to switch.)

b. Run the Gale-Shapley algorithm on the above input. It suffices to state the final matching (the set $S$ of engaged pairs that the algorithm returns).

c. The Gale-Shapley algorithm works in iterations; in every iteration it takes an arbitrary unmatched man and attempts to find his partner. Is it possible to process the men in such order so that the algorithm returns the matching $(A, 4), (B, 3), (C, 1), (D, 2)$? Reason your answer. (If yes, state the order in which the men need to be processed. If not, argue why not.)

Problem 2

This problem is about a puzzle with strings. We are given two strings $u$ and $v$, each of length $n$ and using only lower-case letters. We want to modify $u$ into $v$. The tricky part is that we are allowed to use only this operation: take a contiguous segment of the current string and either

- shift each letter in the segment forward in the alphabet (‘a’ becomes ‘b’, ‘b’ becomes ‘c’, and so on; the segment cannot contain letter ‘z’), or
- shift each letter in the segment backward in the alphabet (‘b’ becomes ‘a’, ‘c’ becomes ‘b’, and so on; the segment cannot contain letter ‘a’).

What is the minimum number of such operations we need to use to modify $u$ into $v$?

**Example 1:** Suppose we want to modify $u =$“hello” into $v =$“teams”. There are several possible ways how modify $u$ into $v$ using 27 operations. For example, we can first shift “lo” forward, getting “helmp”. Then shift “h” forward 12 times, getting “telmp”. Then shift ‘i’ 11 times backward to get “teamp” and then shift “p” forward three times to get “teams”. Total number of operations is $1+12+11+3=27$. It turns out that 27 is the smallest possible number of operations.
Example 2: Suppose we want to modify \( u = \text{“aaccaaaa”} \) into \( v = \text{“bbbbbbbbbb”} \). There are several possible ways how to modify \( u \) into \( v \) using 3 operations. For example, we can first shift the entire string forward, getting “bbdddbbbb”. Then shift “ddd” backward twice to get “bbbbbbbbbb”. This requires 1+2=3 operations.

Your task: Design and implement an \( O(n) \) algorithm that computes the minimum number of operations needed to modify \( u \) into \( v \). In addition to the code,

- submit a pseudo code of your algorithm,
- describe how your algorithm works in plain English (a couple of paragraphs),
- informally argue why your algorithm works (bonus points for formal proof), and
- argue why your algorithm takes \( O(n) \) time.

Note: Polynomial-time algorithms slower than \( O(n) \) are accepted for partial credit. In such case, instead of arguing that your algorithm is \( O(n) \), estimate your algorithm’s running time.

Problem 3

Rank the following functions by order of growth; that is, find an arrangement \( g_1(n), g_2(n), \ldots, g_{24}(n) \) of functions satisfying \( g_i(n) = O(g_{i+1}(n)) \) for every \( i \in \{1, \ldots, 23\} \). Partition your list into equivalence classes such that \( f(n) \) and \( g(n) \) are in the same class if and only if \( f(n) = \Theta(g(n)) \). You do not have to prove your answers.

\[
\begin{align*}
&n \log n & n^{2/3} & n^{3/2} & n^{1/\log n} & \log n & 1 & (\log n)^{\log n} & 2^n \\
&\log_{10} n & 4^{\log n} & n & 2^n & 2^{n+1} & \log \log n & n^{\log \log n} & n! \\
&2^{2^n} & 2^{\log n} & \log^2 n & \log(n^2) & \sqrt{2^{\log n}} & \sqrt{\log n} & n2^n & n + n^2/10^{20}
\end{align*}
\]

Remarks:

- In this class we use \( \log n \) to denote the logarithm base 2.
- Use the Stirling’s formula to figure out how to rank \( n! \). The Stirling’s formula is:

\[
n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + O\left(\frac{1}{n}\right))
\]

- Use also this fact: for any constants \( b_1, b_2 > 0 \):

\[
\log^{b_1} n = O(n^{b_2}) \quad \text{and} \quad n^{b_2} \neq O(\log^{b_1} n)
\]

In words, logarithm of \( n \) raised to any power grows slower than any power of \( n \).